VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Simulátor digitálního teploměru s grafickým rozhraním

Digital Thermometer Simulator with GUI

2014 Barbora Švajcová

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Barbora Švajcová**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Simulátor digitálního teploměru s grafickým rozhraním
Digital Thermometer Simulator with GUI

Zásady pro vypracování:

Navrhně programový simulátor obvodu DS1621. Vytvořený program bude nahrazovat fyzickou součástku programově a bude možno s ním komunikovat přes rozhraní I2C.

1. Seznamte se s komunikačním rozhraním I2C.
2. Navrhněte stavový automat simulující chování podřízeného I2C zařízení.
3. Programově realizujte navržený automat.
4. Pomocí navrženého programového rozhraní realizujte simulaci digitálního teploměru.
5. Navhněte GUI pro monitorování a ovládání teploměru.
6. Zhodnoťte spolehlivost a stabilitu navrženého řešení a porovnejte chování se skutečným teploměrem.

Seznam doporučené odborné literatury:

[1] I2C Specifikace: http://www.nxp.com/documents/application_note/AN10216.pdf
[2] Datový list digitálního teploměru DS1621.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Petr Olivka**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.*

V Ostravě 7. května 2014 .....

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 7. května 2014 .....

Ráda bych na tomto místě poděkovala všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla, především vedoucímu diplomové práce panu Ing. Petru Olivkovi.

Abstrakt

Diplomová práce má za úkol vytvořit programový simulátor obvodu digitálního teploměru Dallas 1621 s grafickým rozhraním. Jedná se o programovou simulaci fyzické součástky teploměru a sběrnice I^2C, pomocí níž teploměr komunikuje s uživatelem. Dále je vytvořen program, který se simulátorem komunikuje a bude sloužit studentům k testování teploměru a jeho vlastností. Komunikace sběrnice I^2C a digitálního teploměru je podrobně popsána stavovými automaty.

Klíčová slova: sběrnice I^2C, digitální teploměr Dallas 1621, stavový automat, GUI aplikace

Abstract

The thesis deals with the creation of digital thermometer circuit Dallas 1621 simulator with grafic user interface. This is a programmatic simulation of physical thermometer component and I^2C bus. Using the bus can thermometer communicate with user. There was also created program, which communicate with simulator and it will serve students for testing thermometer and his properties. Bus and digital thermometer communication is described in detail by state machines.

Keywords: I^2C bus, digital thermometer Dallas 1621, state machine, GUI application

Seznam použitých zkratek a symbolů

DS1621	–	Obvod digitálního teploměru Dallas 1621
I^2C	–	Inter Integrated Circuit Bus, obousměrná dvouvodičová sběrnice
STA	–	Serial DAta, datová linka
SCL	–	Serial CLock, hodinový signál
ACK	–	Acknowledge, potvzení o přijetí bajtu
GTK+	–	Grafická knihovna
GUI	–	Grafic user interface, grafické uživatelské rozhraní
APPS	–	Předmět Architektury počítačů a paralelních systémů
TCP/IP	–	Primární transportní protokol (TCP), protokol síťové vrstvy (IP)

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Diplomová práce se zabývá návrhem a tvorbou programového simulátoru obvodu DS1621. Má za úkol nahradit fyzickou součástku v počítači programově. Aby bylo možné s obvodem komunikovat, musí být naprogramována i komunikační I^2C sběrnice. Výsledný program simulátoru, ve formě GUI aplikace, bude poté použit jako podklad pro výuku v předmětu Architektury počítačů a paralelních systémů [4][1]. Předmět je vyučován na Fakultě elektrotechniky a informatiky Vysoké školy Báňské v Ostravě.

V současné době studenti v rámci výuky programují mikropočítač Atmel ATmega32 v Atmel Studiu. Pomocí sběrnice I^2C se spojí s obvodem DS1621[2] a mohou s ním komunikovat. K otestování vytvořeného programu potřebují fyzickou součástku napojenou přes USB port. Tato skutečnost je důvodem vzniku simulátoru, který součástky nahradí a studenti tak budou moci jednoduše testovat vytvořené programy.

Součástí této práce je podrobné prostudování komunikační I^2C sběrnice, která je základem komunikace s obvodem teploměru. Přehledné řešení funkcionality sběrnice a teploměru je popsáno pomocí tří stavových automatů. Tyto automaty slouží jako podklad pro programovou realizaci simulátoru.

Nedílnou součástí projektu je také jednoduché grafické uživatelské rozhraní simulátoru. K aplikaci simulátoru se přes TCP/IP port připojí aplikace klienta. Do klientské části uživatelé vkládají své testovací funkce. Komunikace zde probíhá pomocí socketů, které jsou převzaty z diplomové práce pana Michala Fůse[3].

Aplikace simulátoru je spustitelná v operačním systému Windows.

V závěru práce je popsáno srovnání chování a ovládání fyzické součástky digitálního teploměru a jeho programové simulace. Byla provedena řada testů funkcionality i srovnávacích testů. Průběh a výsledky jsou podrobně popsány v kapitole 9.

[1] dále jen APPS

2 Zadání a specifikace požadavků

2.1 Současný stav

V současné době je při výuce používán obvod digitálního teploměru DS1621 pro spojité měření teploty, který je připojen na sběrnici I^2C. Studenti programují mikropočítač Atmel ATmega32 pomocí vývojového prostředí Atmel Studio. Výsledný kód umožňuje komunikovat s připojeným teploměrem a nastavovat jeho konfiguraci programově.

2.2 Specifikace programového simulátoru

Vytvořit programový simulátor obvodu DS1621, který fyzickou součástku teploměru nahradí, včetně jednoduchého uživatelského rozhraní.

2.2.1 Funkční požadavky

- zobrazení a změna aktuální teploty,
- zobrazení poslední naměřené teploty,
- zobrazení aktuální stavu termostatu (zapnuto/vypnuto),
- zobrazení průběhu měření (měří/neměří),
- zobrazení druhu měření (spojité/jednotlivé),
- zobrazení a změna konfiguračního registru teploměru,
- zobrazení a změna adresy teploměru,
- zobrazení a změna horního a dolního prahu teploměru,
- po ukončení spojení s klientem zůstane simulátor nastaven poslední známou konfigurací,
- po ukončení aplikace bude poslední nastavení simulátoru uloženo do konfiguračního souboru,
- po spuštění bude aplikace nakonfigurována z konfiguračního souboru,
- klient bude využívat knihovnu, určenou pro komunikaci se simulátorem, kde bude moci zasílat i přijímat data a následně je vypisovat na konzoli klienta.

2.2.2 Nefunkční požadavky

- operační systém Windows,
- GUI aplikace termostatu,
- konzolová aplikace klienta obsahující knihovny pro komunikaci přes I^2C sběrnici.

3 Komunikační rozhraní I^{2}C

3.1 Používaná terminologie

Transmitter - zařízení, které odesílá data na sběrnici. Vysílač může být také zařízení, které „začalo“ komunikaci, master zařízení, nebo to, které na základě požadavku master zařízení zasílá požadovaná data (slave-transmitter).

Receiver - zařízení, které přijímá data ze sběrnice.

Master - komponenta, která iniciuje přenos dat, generuje hodinový signál a ukončuje přenos. Master zařízení může být buď vysílač nebo přijímač.

Slave - zařízení adresované zařízením typu master. Slave zařízení může být buď vysílač nebo přijímač.

Multi-master - znamená schopnost spoluexistence více zařízení master na jedné sběrnici ve stejném čase bez kolizí a ztráty přenášených dat.

Arbitráž - předpřipravená procedura, která autorizuje jen jedno zařízení master, aby zajistilo kontrolu nad sběrnicí.

Synchronizace - hodinový signál poskytnutý jedním nebo více zařízeními typu master.

SDA - Serial DAta, datová linka.

SCL - Serial CLock, linka hodin.

3.2 O komunikačním rozhraní I^{2}C

[1] Sběrnice I^{2}C byla navržena firmou Philips Semiconductors pro komunikaci jednočipových mikropočítačů s dalšími číslicovými obvody.

I^{2}C je obousměrná dvouvodičová sběrnice. Někdy bývá ale nesprávně označována jako sériová linka. Zkratka I^{2}C označuje *Inter Integrated Circuit Bus*. Český překlad by mohl být meziobvodová sběrnice.

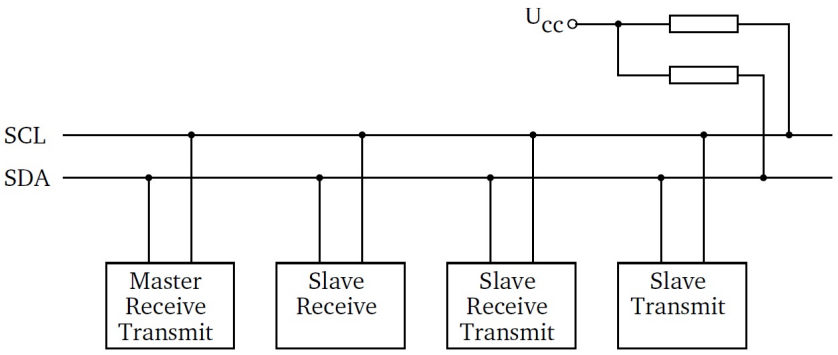
3.2.1 Základní charakteristika

Sběrnice se skládá ze dvou linek. SDA slouží pro obousměrný přenos dat a SCL linka slouží jako hodinový signál. Obě linky pracují v napěťových logických úrovních shodně s technologií TTL.

V klidovém stavu jsou obě linky v úrovni log. 1. Tento stav je udržovaný dvěma zdvihacími (pull-up) rezistory. Výstupy číslicových obvodů jsou obvykle řešeny jako výstup typu otevřený kolektor. Tím je zaručena obousměrnost linky.

Základní uspořádání systému můžeme vidět na obrázku č. 1. Na sběrnici může být připojeno více zařízení. Obvykle jedno zařízení označované jako master řídí vysílání a příjem zpráv. Další zařízení, označovaná jako slave, po výzvě nadřízeným obvodem data

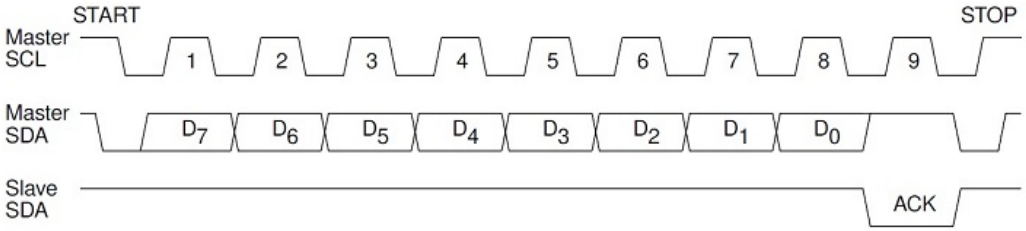
přebírají (receive), nebo odesílají (transmit). Na sběrnici smí být i více zařízení typu master. Každé zařízení, které se připojí k téže sběrnici musí mít jedinečnou adresu.



Obrázek 1: Uspořádání typického systému sběrnice.[4]

3.2.2 Přenos bitů

Přenos bitů probíhá na sběrnici synchonně. Synchronizace se provádí hodinovým signálem SCL, který řídí zařízení master. V jednom hodinovém cyklu je přenesen vždy jeden bit. Při datovém přenosu je možné měnit SDA signál jen tehdy, pokud je hodinový signál v log. 0. Pokud by došlo ke změně dat během hodinového signálu v log. 1, pak se vždy jedná o tzv. řídící signál. Mezi tyto patří START (S), STOP (P) a REPEAT (má stejný průběh jako START). Řídící signály smí generovat pouze zařízení master.



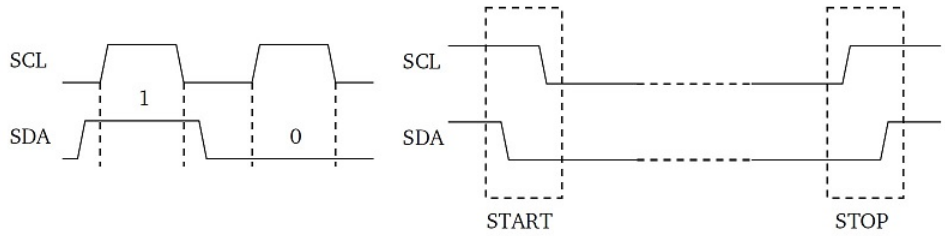
Obrázek 2: Příklad komunikace zařízení master a slave.[4]

3.2.3 Popis průběhů signálů START a STOP

Na počátku každé komunikace musí být vyslán řídící signál START. Počáteční podmínkou je klidový stav sběrnice (SDA i SCL v log. 1). První do log. 0 přechází signál SDA, kdy SCL je stále v log. 1. Jako druhý přechází do log. 0 hodinový signál SCL.

Pro ukončení komunikace se sběrnice musí uvést opět do klidového stavu, tedy SCL i SDA musí být v log. 1. V tomto případě je vyslán řídící signál STOP. Jako první do log. 1 přechází signál SCL a jako druhý přechází do log. 1 signál SDA.

Graficky znázorněné průběhy signálů můžete vidět na obrázku č. 3.



Obrázek 3: Přenos hodnoty 1 a 0. Průběh řídících signálů START a STOP. [4]

3.2.4 Průběh komunikace, obecné podmínky při komunikaci přes I^{2}C sběrnici

Komunikace je ustálená a data jsou přenášena po bajtech. Devátý bit slouží k potvrzování příjmu každého bajtu dat přijímacím zařízením, dokud nejsou přenesena všechna data. Sběrnice se uvolní pro ostatní zařízení, pokud zařízení master přivede linku SDA do log. 1 v průběhu horní úrovně signálu SCL. Vyjma dvou vyjímek START a STOP signálů, není žádné zařízení oprávněno měnit signál SDA, dokud není signál SCL v log 0. Po signálu START je sběrnice považována za zaneprázdněnou. Zůstává zaneprázdněná také ve chvíli, kdy zařízení master generuje opakovaný signál START (REPEAT) místo signálu STOP. Sběrnice se uvolní po signálu STOP.

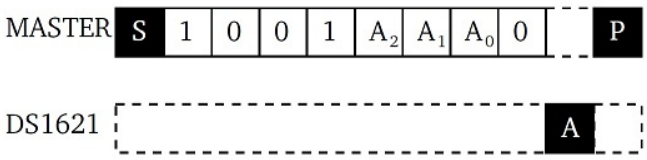
H_3	H_2	H_1	H_0	A_2	A_1	A_0	$R\overline{W}$

Obrázek 4: Formát adresy obvodu digitálního teploměru na I^{2}C.[4]

Jakékoliv I^{2}C zařízení může být připojeno ke společné I^{2}C sběrnici a vzájemně spolu komunikovat, přenášet informace tam a zpět. Každé zařízení má unikátní 7mi bitovou nebo 10ti bitovou adresu [2]. Bity označené na obrázku č. 4 jako H0 - H3 jsou pevně dány (pro teploměr DS1621 jsou to 1001), další tři bity označené jako A0-A2 jsou určeny pro jednoznačnou adresu zařízení. To umožňuje uživateli připojit až osm různých zařízení ke stejné I^{2}C sběrnici. Poslední, osmý bit adresy ($R/\overline{W}$), udává, zda master zařízení bude

[2] V diplomové práci se používá jen 7mi bitová adresa.

vysílat (write) či přijímat (read) data. Každý přenos dat musí začít START sekvencí a končit STOP sekvencí.



Obrázek 5: Vyslání adresy s potvrzením od slave zařízení.[4]

Při osmém hodinovém pulsu je SDA nastaveno buď na log. 1 (read - data budou přečtena z adresovaného slave zařízení) nebo na log. 0 (write - data budou zaslána zařízením typu master adresovanému zařízení). Během devátého hodinového cyklu zařízení master uvolní SDA linku, aby se uskutečnila potvrzovací fáze (acknowledge). Pokud je tedy volané slave zařízení připojeno ke sběrnici, úspěšně dekódovalo a rozeznalo svoji adresu, potvrdí přijetí tak, že SDA linku uvede do log. 0.

Podrobnější průběh komunikace je uveden v kapitole 4.

4 Obvod digitálního teploměru Dallas DS1621

[2]Obvod DS1621 je digitální teploměr, který může pracovat jako teploměr pro spojité měření teploty, provádět měření teploty na požádání, nebo pracovat jako termostat s nastavitelnou hysterezí.

4.1 Popis fyzické součástky

Teplotní rozsah použitého teploměru je -55 °C až +125 °C. Teplota je čtena jako 9ti bitová hodnota, která se posílá vždy ve dvou bajtech. Nastavení teploměru je uživatelsky definovatelné a lze tedy měnit. Data jsou zasílána a čtena pomocí komunikačního rozhraní I^2C sběrnice. Teploměr je vybaven obousměrným sériovým portem pro signál SDA a také hodinovým signálem SCL. Dále obsahuje výstupní signál ve formě LED diody, která reprezentuje aktuální stav termostatu (zapnuto, vypnuto), a tři čipové vstupy určené pro 3 bitovou adresu zařízení.

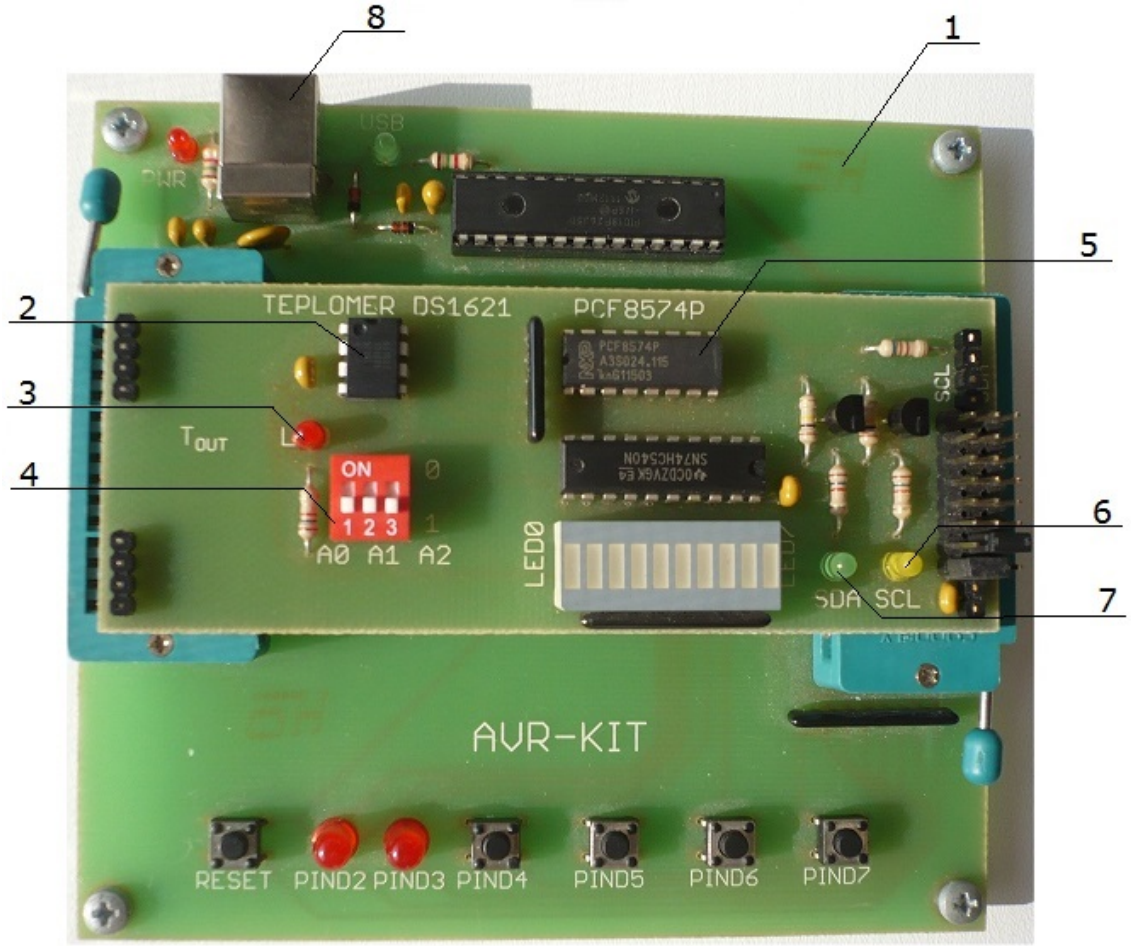
4.1.1 Popis uložení teploty

Teploměr umožňuje čtení naměřené teploty pomocí 9ti bitů. Teplota v teploměru se ukládá do dvou bajtů, které se rozlišují jako MSB[3] a LSB[4]. Ve vyšším MSB bajtu je uložena celočíselná část teploty, v dolním LSB bajtu se uchovává pouze rozlišení teploty o půl stupně. K tomuto rozlišení se využívá pouze nejvyšší bit druhého bajtu, ostatní jsou vždy nastaveny na 0. V tabulce na obrázku č. 6 jsou příklady uložení teploty.

TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125°C	01111101 00000000	7D00h
+25°C	00011001 00000000	1900h
+½°C	00000000 10000000	0080h
+0°C	00000000 00000000	0000h
-½°C	11111111 10000000	FF80h
-25°C	11100111 00000000	E700h
-55°C	11001001 00000000	C900h

Obrázek 6: Příklady uložení teploty v paměti teploměru.[2]

[3]Most Significant Bit
[4]Least Significant Bit



Obrázek 7: Obvod digitálního teploměru DS1621 s modulem AVR-KIT.
Popis : 1 - modul AVR-KIT, **2** - teploměr DS1621, **3** - zobrazení stavu termostatu (svítí - zapnuto, nesvítí - vypnuto), **4** - přepínače pro nastavení adresových bitů teploměru, **5** - 8 bitový expandér PCF8574, **6** - zobrazení činnosti SCL linky, **7** - zobrazení činnosti SDA linky, **8** - zdířka pro připojení USB kabelu

4.1.2 Obsah konfiguračního registru

Důležitou součástí teploměru je konfigurační registr (viz. obrázek č. 8), kde jsou uložena nastavení termostatu. Registr je 8mi bitový a je definován následovně :

MSb	Bit 6	Bit5	Bit 4	Bit 3	Bit 2	Bit 1	LSb
DONE	THF	TLF	NVB	X	X	POL	1SHOT

Obrázek 8: Konfigurační registr teploměru.[2]

DONE – 0 probíhá konverze teploty, 1 konverze dokončena.

THF – Temperatur High Flag, tento příznak je nastaven na 1 pokud je měřená teplota větší nebo rovna horní hranici.

TLF – Temperature Low Flag, tento příznak je nastaven na 1 pokud je měřená teplota menší nebo rovna dolní hranici.

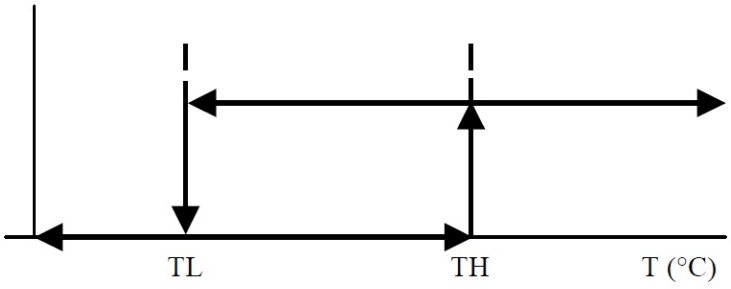
NVB – Nonvolatile Memory Bus, 1 – probíhá zápis do paměti, paměť je zaneprázdněna.

POL – Output Polarity Bit, 1 – aktivní v log 1 (active high), 0 – aktivní v log 0 (active low).

1SHOT – One Shot Mode, 1 – jednotlivé měření teploty, 0 – spojité měření teploty.

X – rezervované bity.

Spínání termostatu při polarizaci „active high“ je graficky znázorněno na obrázku č. 9. Termostat sepne ve chvíli, kdy teplota přesáhne uživatelem definovanou horní hranici TH. Termostat zůstává sepnutý až do chvíle, kdy teplota dosáhne uživatelem definované dolní hranice TL a to bez ohledu na hysterezi.



Obrázek 9: Spínání při polarizaci „active high“.[2]

Nastavení teploměru jsou ukládána do stálé paměti, takže lze termostat předprogramovat před jeho zapojením do systému. Veškerá nastavení a teploty jsou zasílané pomocí dvou vodičového sériového rozhraní.

Horní i dolní teplotní hranice jsou uloženy zvlášť v paměti teploměru a jsou uživatelsky definovatelné.

4.2 Příkazy pro ovládání teploměru

V prvním bajtu zařízení master při komunikaci s teploměrem zasílá adresu volaného zařízení a bit $R/\overline{W}$[5]. Druhý bajt obsahuje jeden ze sady příkazů, určených pro teploměr. Podrobný popis komunikace je k dispozici manuálu[2], graficky je znázorněn také v příloze 19. Mezi nejpoužívanější příkazy patří :

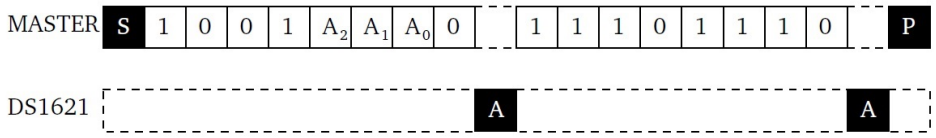
AAh (Read Temperature) – zařízení master požaduje po teploměru zaslání poslední naměřené teploty (dva bajty).

ACh (Access Config) – tento příkaz umožňuje přístup ke konfiguračnímu registru teploměru, pokud je bit $R/\overline{W} = 0$, pak zařízení master zapisuje data do konfiguračního registru, pokud je bit $R/\overline{W} = 1$, pak teploměr zasílá aktuální registr zařízení master.

EEh (Start Convert T) – tímto příkazem se spouští měření a převod teploty (spojité i jednotlivé), viz. obrázek č. 10.

22h (Stop Convert T) – konec měření a převodu teploty.

Dále jsou předmětem komunikace například nastavení či čtení horního a dolního teplotního limitu (A1h, A2h).



Obrázek 10: Příkaz Start Convert T, ke spuštění převodu teploty.[4]

4.2.1 Průběh komunikace

Komunikace teploměru se zařízením master může probíhat pěti základními způsoby. Každá komunikace se zařízením, jak bylo zmíněno výše, začíná příkazem START, poté následuje 7mi bitová adresa slave zařízení a bit $R/\overline{W}$. Po přijetí prvních 8mi bitů vyšle volané slave zařízení potvrzení ACK (acknowledge). V dalších 8mi bitech zařízení master zasílá command byte (příkaz). Slave zařízení jej po přijetí posledního bitu zpracuje a zašle opět potvrzení ACK.

[5] $\mathrm{Read}/\overline{\mathrm{Write}}$, viz. kapitola 3.2.4

Výše popsaným způsobem začíná všech pět základních druhů komunikace, popsaných v dokumentaci digitálního teploměru DS1621[2].

Send "standalone"command - další data nezasílá a komunikaci ukončuje zařízení master příkazem STOP. Používá se pro příkazy ke spuštění či ukončení digitalizace teploty.

Write to a single-byte register - master zařízení zasílá ještě jeden bajt a po ACK od slave zařízení ukončí komunikaci. Používá se pro zápis do konfiguračního registru.

Write to a two-byte register - master zařízení zasílá dva bajty (každý přijatý bajt potvrdí slave zařízení potvrzením ACK), pak je ukončena komunikace. Používá se pro nastavení horní a dolní hranice termostatu.

Read from a single-byte register - master zařízení vyšle signál REPEAT a znovu zasílá na sběrnici adresu volaného zařízení. Změnou je poslední bit $R/\overline{W}$, který je nyní nastaven na Read (log. 1). Zařízení slave vyšle potvrzení ACK a následně požadovaný bajt. Po přijetí posledního bitu zašle master zařízení potvrzení NACK a ukončí komunikaci. Používá se pro čtení dat konfiguračního registru (dále také pro příkazy slope či counter, které v této práci nejsou používány).

Read from a two-byte register - průběh komunikace je stejný jako čtení jednoho bajtu, jen mezi prvním a druhým bajtem zasílaným od slave vyšle master zařízení potvrzení ACK. Používá se pro čtení naměřené teploty a horní a dolní hranice termostatu.

Podrobnější informace, včetně přehledného obrázku č. 19, naleznete v příloze nebo v manuálu teploměru[2]. Komunikace byla také naprogramována v rámci testování simulátoru. Funkce ve zdrojovém kódu jsou uvedeny v příloze B.2.

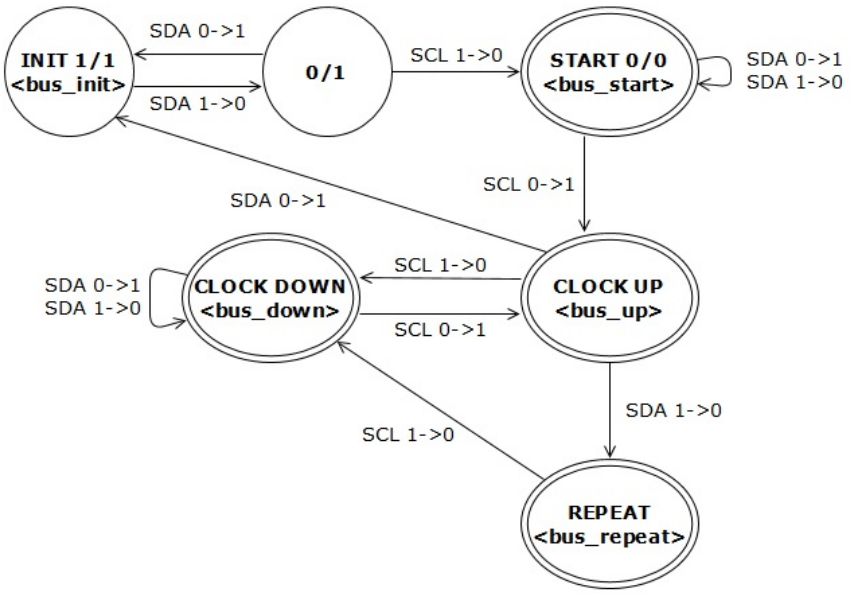
5 Postup návrhu stavových automatů I²C

V další části je popsán postup při návrhu konkrétních stavových automatů, určených ke komunikaci I²C sběrnice a digitálního teploměru. Návrh se skládá ze tří vrstev : fyzická vrstva, vrstva I²C automatu a vrstva I²C zařízení, které simulují vzájemnou komunikaci mezi zařízeními master a slave. Pro každou vrstvu byl navrhnut stavový automat.

První, fyzická vrstva, má za úkol zpracovat signály SDA a SCL. Ve druhé vrstvě, I²C automatu, jsou již rozpoznávány konkrétní stavy komunikace na sběrnici (čtení adresy, zápis bajtu apod.) a ve třetí vrstvě, I²C zařízení, je již reprezentováno připojené komunikační zařízení (digitální teploměr). Programový simulátor je tedy postaven na třívrstvé architektuře, vzájemně propojené událostmi.

5.1 První vrstva - fyzická

Ve fyzické vrstvě jsou podrobně sledovány signály SCL a SDA. Navržený stavový automat reaguje na každou změnu těchto signálů. Jak je vidět na obrázku č. 11, jsou pomocí toho automatu rozeznávány stavy - Init, 0/1, Start, Clock up, Clock down a Repeat. Na základě těchto stavů jsou generovány události, které jsou zasílány druhé vrstvě - I²C automatu, ke zpracování.



Obrázek 11: Stavový automat I²C Sběrnice. **Popis :** Přechody mezi stavy zde určují měnící se hodnoty na linkách SDA a SCL, které jsou při inicializaci shodně v log. 1. Stavy automatu jsou označeny velkými písmeny, generované události jsou označené <bus_event_name>. Stavy, které jsou ve dvojitém kruhu generují události, které se přenáší do druhé vrstvy.

Události fyzické vrstvy, které jsou zasílány druhé vrstvě :

- **bus_start**,
- **bus_up**,
- **bus_down**,
- **bus_repeat**.

5.2 Druhá vrstva - I²C automat

Ve druhé vrstvě jsou zachyceny události první vrstvy. Dle jejich druhu a aktuálního stavu I²C automatu jsou dále zpracovány (viz. obrázek č. 12). Automat již rozlišuje stavy jako čtení adresy, čtení nebo zápis bajtu, čtení nebo zápis příkazu. Dle svého aktuálního stavu automat rozhodne, zda je změnu nutné oznámit třetí vrstvě (teploměru) či nikoliv. Pokud ano, pak vygeneruje novou událost, která je odeslána ke zpracování třetí vrstvě. Automat také počítá přenesené bity a po každých osmi přenesených bitech oznamuje, že je očekáván potvrzovací bit (acknowledge).

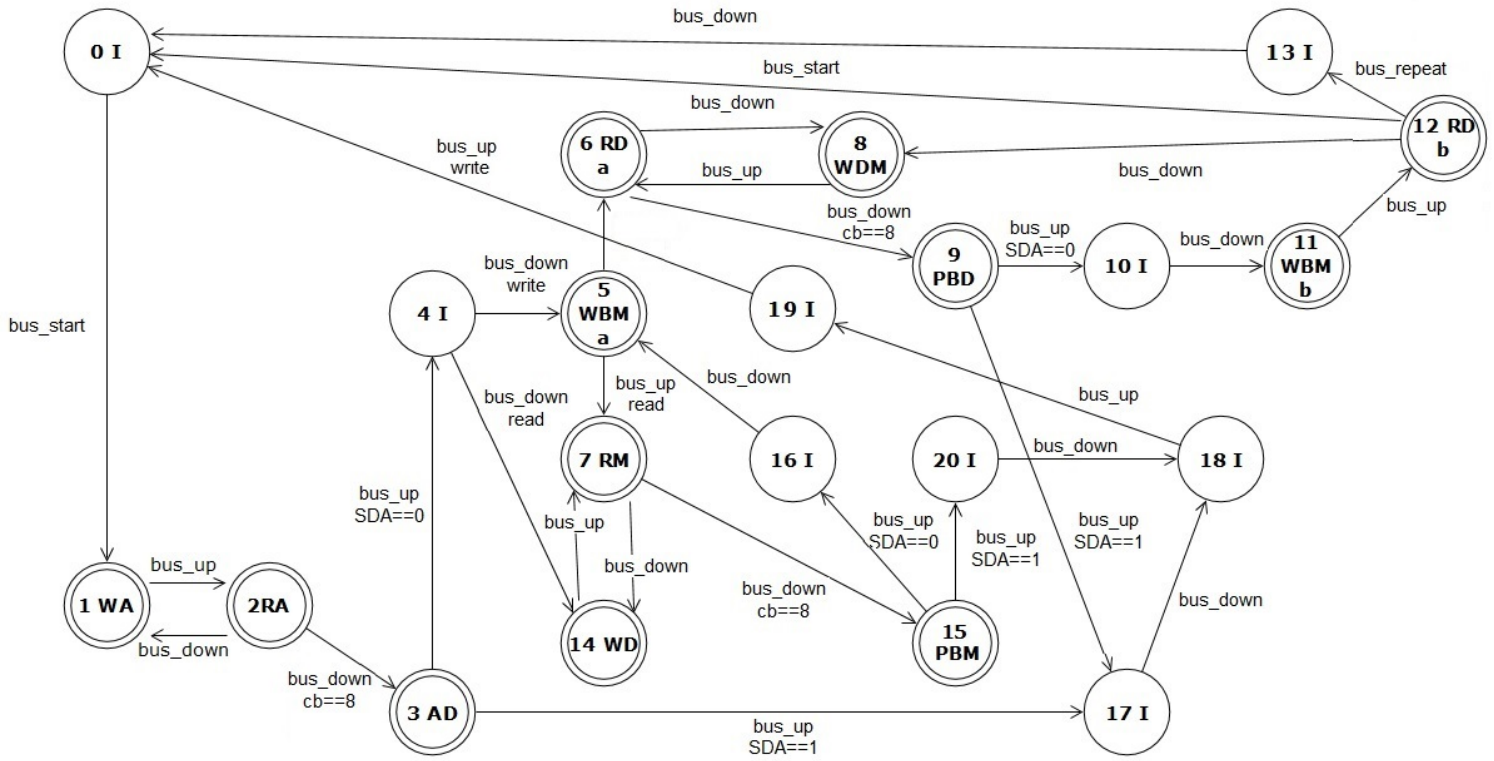
Automat je graficky znázorněn na obrázku č. 12. Následuje podobný popis jednotlivých stavů automatu[6] pro zpracování I²C událostí[7] :

- **0 - Start** - inicializace všech zařízení, countByte = 0, <aut_internal>.
- **1 - Write_addr** - master zařízení zasílá adresu včetně bitu $R/\overline{W}$, countByte++, <aut_write_addr>.
- **2 - Read_addr** - všechny slave zařízení čtou data z linky SDA, <aut_read_addr>.
- **3 - Address** - slave zařízení kontroluje adresu, zjišťuje bit $R/\overline{W}$ a zasílá potvrzení na linku SDA, <aut_address>.
- **4 - Ack_device** - slave zařízení je připraveno přijmout data, <aut_internal>.
- **5 - Write_bit_master** - countByte = 0, dle bitu $R/\overline{W}$ se zvolí hrana a vysílač (zařízení master nebo slave) nastaví první bit, countByte++, <aut_write_bit_master>.
- **6 - Read_device** - slave zařízení čte bit, <aut_read_device>.
- **7 - Read_master** - master zařízení čte bit, <aut_read_master>.
- **8 - Write_data_master** - master zařízení nastavuje další bit, countByte++, <aut_write_data_master>.

[6] countByte zde slouží jako čítač poslaných bitů.

[7] Události, které stavy generují, jsou označeny <aut_event_name>, událost aut_internal není třetí vrstvě oznamována.

- **9 - Process_byte_device** - kontrola přijatého bajtu (pokud ještě nebyl přijat command byte, pak je nyní identifikován, jinak dle aktuálního příkazu se provádí zápis přijaté hodnoty), pokud je v pořádku, nastaví slave na linku SDA potvrzení o přijetí bajtu, log. 0, <aut_process_byte_device>.
- **10 - Send_out_or_stop** - kontrola, zda již není konec vysílání, countByte++, <aut_internal>.
- **11 - Write_bit_master2** - master zařízení nastaví první datový bit nebo následuje signál REPEAT, countByte++, <aut_write_bit_master2>.
- **12 - Read_device2** - slave zařízení čte první bit nebo konec je vysílání, <aut_read_device2>.
- **13 - Repeat** - generován signál REPEAT, <aut_internal>.
- **14 - Write_device** - slave zařízení nastavuje další bit, <aut_write_device>.
- **15 - Process_byte_master** - master zařízení provádí kontrolu přijatého bajtu, dle toho nastaví SDA linku na 0 - ACK, 1 - NACK, <aut_process_byte_master>.
- **16 - Ack_master** - slave zařízení zjistí, zda zařízení master přijal bajt a jde vysílat další, <aut_internal>.
- **17 - Nack_device** - slave zařízení nebylo nalezeno nebo neodpovídá, <aut_internal>.
- **18 - Nack_master** - zařízení master přijalo poslední bajt a bude ukončeno vysílání, <aut_internal>.
- **19 - Wait_for_stop_1** - čekání na konec vysílání, <aut_internal>.
- **20 - Wait_for_stop_2** - čekání na konec vysílání, <aut_internal>.



Obrázek 12: Stavový automat pro zpracování I^{2}C událostí bus_event_name. Stavy jsou číslované. Písmena označují generované události. Stavy ve dvojitém kruhu generují události, které se přenáší do třetí vrstvy, cb - countByte (počet přijatých bitů). **Popis stavů**: **0** - Start, **1** - Write_addr, **2** - Read_addr, **3** - Address, **4** - Ack_device, **5** - Write_bit_master, **6** - Read_device, **7** - Read_master, **8** - Write_data_master, **9** - Process_byte_device, **10** - Send_out_or_stop, **11** - Write_bit_master2, **12** - Read_device2, **13** - Repeat, **14** - Write_device, **15** - Process_byte_master, **16** - Ack_master, **17** - Nack_device, **18** - Wait_for_stop1, **19** - Wait_for_stop2, **20** - Nack_master. **Popis událostí**: **I** - aut_internal, **WA** - aut_write_addr, **RA** - aut_read_addr, **AD** - aut_address, **WBMa** - aut_write_bit_master, **RDa** - aut_read_device, **RM** - aut_read_master, **WDM** - aut_write_data_master, **PBD** - aut_process_byte_device, **WBMb** - aut_write_bit_master2, **RDb** - aut_read_device2, **WD** - aut_write_device, **PBM** - aut_process_byte_master.

5.3 Třetí vrstva - digitální teploměr

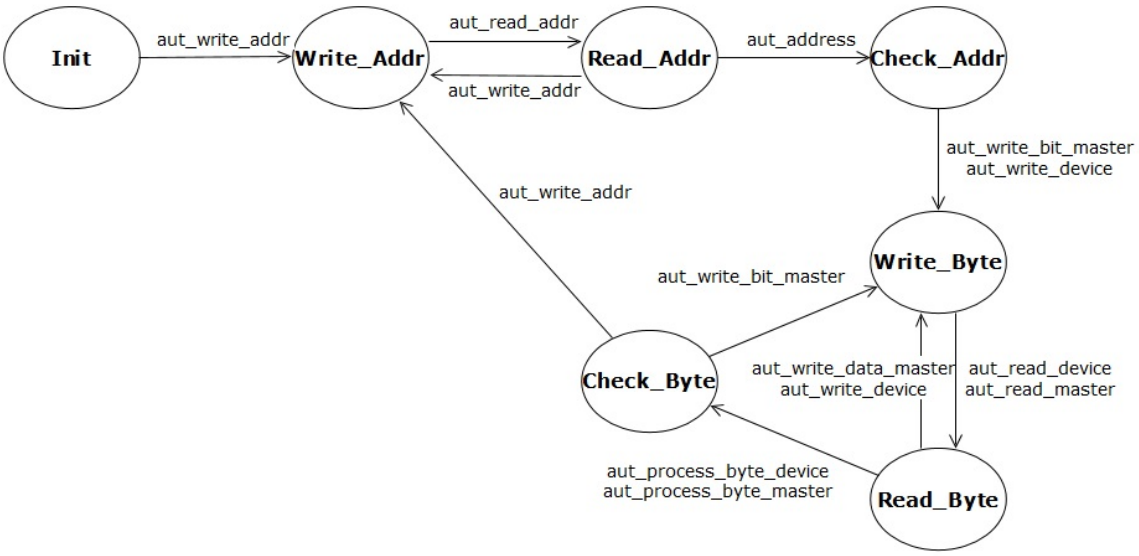
Ve třetí vrstvě již na události ze druhé vrstvy reaguje přímo připojené slave zařízení (v našem případě teploměr) s ohledem na svůj aktuální stav. Data zpracovává až po přijetí osmého bitu.

Pokud teploměr zasílá data na sběrnici, pak požadovaná data nejprve uloží do speciálního registru, který je určen pro odesílání. Po osmi bitech očekává od master zařízení potvrzovací bit.

Třetí vrstva generuje také události, které budou sloužit ke komunikaci s konzolovou aplikací, či vytvořeným grafickým uživatelským rozhraním. Komunikace zde probíhá oběma směry, takže teploměr reaguje na změny v GUI opět na základě událostí, generovaných samotnou aplikací. Tímto se zabývá podrobně kapitola č. 7 „Návrh GUI".

Následuje výpis událostí druhé vrstvy a jejich zpracování ve třetí vrstvě. Vzhledem k tomu, že zařízení master i slave používají ke čtení i zápisu stejné funkce, slouží k jejich rozlišení proměnná isMaster. Pokud je její hodnota 0, pak čte či zapisuje zařízení slave, pokud je její hodnota 1, pak čte či zapisuje zařízení master. Proměnná countByte opět slouží jako počítadlo odeslaných a přijatých bitů.

- **aut_write_addr** - neprobíhá další zpracování, data zasílá zařízení master,
- **aut_read_addr** - jestliže je countByte < 8, pak je volána funkce Read_Addr(),
- **aut_address** - volání fce pro kontrolu adresy Check_Address(), nastavení potvrzovacího bitu acknowledge,
- **aut_write_bit_master, aut_write_bit_master2** - jestliže je $R/\overline{W}$==1, pak isMaster = 0 jinak isMaster = 1, poté se vždy zavolá fce Write_Byte(),
- **aut_read_device, aut_read_device2** - jestliže countByte < 8, pak isMaster = 0 a volá se funkce Read_Byte(),
- **aut_read_master** - jestliže countByte < 8, pak isMaster = 1 a volá se funkce Read_Byte(),
- **aut_write_data_master** - isMaster = 1,
- **aut_process_byte_device** - isMaster = 0, volá se funkce Check_Byte(), kde je dále rozpoznán bajt, který obsahuje command nebo data, nastavuje se potvrzovací bit acknowledge,
- **aut_write_device** - isMaster = 0, volá se funkce Write_Byte(),
- **aut_process_byte_master** - isMaster = 1, volá se funkce Check_Byte(), nastavuje se potvrzovací bit acknowledge.



Obrázek 13: Stavový automat I^2C zařízení. **Popis :** Stavy jsou volané funkce teploměru, přechody mezi stavy jsou události druhé vrstvy.

6 Programová realizace stavových automatů

Programová realizace vytvořených stavových automatů je postavena na navržené třívrstvé architektuře, která je popsána v kapitole č. 5. Každý automat má definovánu svoji posloupnost stavů a událostí. Pro každý automat byli naprogramovány funkce pro zpracování jednotlivých stavů a také byly programově realizovány funkce pro předávání aktuálních události mezi jednotlivými vrstvami. Při každé změně stavu některého z automatů se aktualizuje jeho současný stav. Pokud je nutné změnu stavu oznámit další vrstvě, jsou vygenerovány nové události. Ty se dále zpracovávají ve funkcích volaného automatu.

Dále byli definovány pro každou vrstvu další vnitřní proměnné, které pomáhají jednoznačně určit do jakého stavu má automat přejít a jaké události generovat.

Mimo jiné byl do aplikace zapracován také uživatelský program, určený k ovládání sběrnice I^2C, který se v současné době používá při výuce v předmětu APPS [4]. Funkce, které se dříve používaly ke komunikaci s připojenou sběrnicí, byli nahrazeny funkcemi simulátoru. Konkrétně se jedná pouze o zápis a čtení signálů z linek SDA a SCL. Takže změna ve zdrojovém kódu by neměla být žádná.

```
// nastaveni signalu SDA
void I2C_SDA( char value )
{
  if ( value ) I2C_PORT |= ( 1 << SDA_PIN );
  else         I2C_PORT &= ~( 1 << SDA_PIN );
  I2C_delay();
}

// cteni signalu SDA
char I2C_getSDA( void )
{
  char r = ( I2C_PIN & ( 1 << SDA_PIN ) ) != 0;
  I2C_delay();
  return r;
}

// nastaveni signalu SCL
void I2C_SCL( char value )
{
  if ( value ) I2C_PORT |= ( 1 << SCL_PIN );
  else         I2C_PORT &= ~( 1 << SCL_PIN );
  I2C_delay();
}
```

Výpis 1: Původní funkce pro zápis a čtení linek SDA a SCL.

```
// nastaveni signalu SDA
void I2C_SDA( char value )
{
    Bus_Set_SDA(value);
}

// cteni signalu SDA
char I2C_getSDA( void )
{
    return Bus_Get_SDA();
}

// nastaveni signalu SCL
void I2C_SCL( char value )
{
    Bus_Set_SCL(value);
}
```

Výpis 2: Nahrazení zápisu a čtení linek SDA a SCL novou funkcí.

Konfigurace aplikace simulátoru je při každém spuštění nastavena dle konfiguračního souboru. Při prvním spuštění jsou v souboru nastaveny implicitní hodnoty. Při ukončení práce s aplikací je aktuální nastavení simulátoru uloženo zpět do konfiguračního souboru.

```
address = 158; //adresa teplomeru
temp_MSB = 255; //posledni namerena teplota
temp_LSB = 128;
high_temp_MSB = 120; //horni teplotni limit
high_temp_LSB = 0;
low_temp_MSB = -55; //dolni teplotni limit
low_temp_LSB = 0;
act_temp = 22; //aktualni teplota teplomeru ve stupnich
config = 131; //konfiguracni registr
slope = 0;
counter = 0;
digitalization = 0; //0 - neprobiha prevod teploty, 1 - probiha prevod teploty
```

Výpis 3: Implicitní nastavení konfiguračního souboru simulátoru digitálního teploměru.

Na konci této fáze projektu již byl realizován první funkční simulátor digitálního teploměru, který komunikoval pouze pomocí konzolové aplikace. V této fázi se doladila veškerá vnitřní komunikace přes sběrnici I^2C. Po dopracování všech detailů začíná další práce na GUI (kapitola 7) a také velice důležitá synchronizace komunikace mezi zařízeními master a slave pomocí socketů a vláken podrobně vysvětlená v kapitole 8.

Následuje výpis ze zdrojového kódu aplikace. Jsou zde zobrazeny všechny paměťové proměnné digitálního teploměru. Tato struktura je v programu využita pro master i slave zařízení (pro jejich rozlišení slouží první proměnná isMaster). Master zařízení ovšem aktivně používá jen prvních sedm proměnných vyjma myAddress.

```
#define DEV_COMMAND_TYPE { DEV_START_CONV, DEV_STOP_CONV,
    DEV_WRITE_CONFIG, DEV_WRITE_TH, DEV_WRITE_TL, DEV_READ_CONFIG,
    DEV_READ_SLOPE, DEV_READ_COUNT, DEV_READ_TH, DEV_READ_TL,
    DEV_READ_TEMP, DEV_NULL }

typedef enum DEV_COMMAND_TYPE dev_command_type;

typedef struct {
    char isMaster; // 0 - slave, 1 - master
    unsigned char myAddress; // 7mi bitova adresa zarizeni
    unsigned char readAddress; // prectena adresa ze sbernice
    unsigned char myByte; // co bude odeslano
    unsigned char myByteLSB;
    unsigned char readByte; // co bylo prijato
    unsigned char readByteMSB;
    unsigned char temperatureMSB; // aktualni teplota
    unsigned char temperatureLSB;
    unsigned char highTemperatureMSB; // horni hranice termostatu
    unsigned char highTemperatureLSB;
    unsigned char lowTemperatureMSB; // dolni hranice termostatu
    unsigned char lowTemperatureLSB;
    unsigned char configRegister; // konfiguracni register
    unsigned char slope;
    unsigned char counter;
    dev_command_type command; // typ prikazu
    char R_W;
    char wait_stop;
    char polarity ;
    char oneShot;
    char *mistake;
    char thermostatOn;
    }  Device;
```

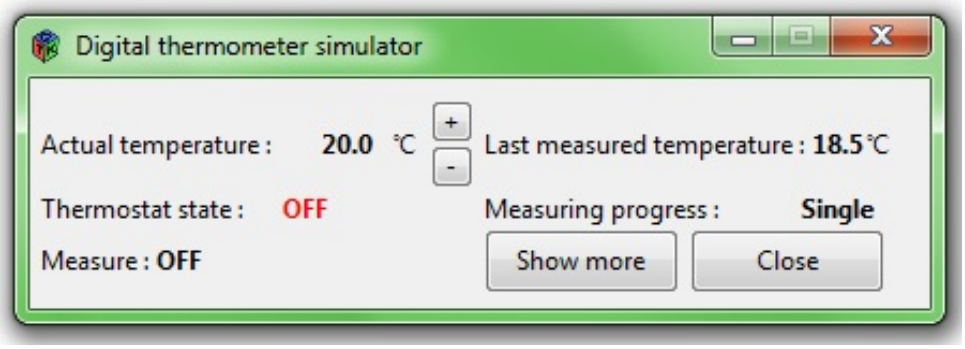
Výpis 4: Vnitřní proměnné teploměru.

7 Návrh grafického uživatelského rozhraní

Dalším krokem byl návrh jednoduchého grafického uživatelského rozhraní[8], kde mohou studenti snadno nastavovat a monitorovat chování teploměru. Jeden z požadavků byl, aby okno aplikace bylo co možná nejmenší a obsahovalo pouze základní údaje o termostatu. Proto bylo realizováno jen malé rozhraní s možností zobrazení podrobných informací, které jsou dostupné po stisku tlačítka.

Navržené GUI je uživatelsky velmi nenáročné. Slouží k jednoduchému sledování simulátoru a umožňuje monitorovat průběh komunikace se zařízením. V GUI mohou studenti nastavit aktuální teplotu a adresu teploměru. Všechny ostatní údaje slouží pouze jako informace o aktuálním stavu připojeného teploměru, ty se mění v závislosti na jejich programovém nastavení.

Hlavní okno aplikace je složeno ze dvou částí. První část obsahuje základní údaje o termostatu. Po odkliknutí tlačítka "Show more"jsou k dispozici další podrobnější údaje o aktuálním nastavení teploměru jako například konfigurační registr, horní a dolní hranice termostatu a adresa teploměru (bity A2 - A0).



Obrázek 14: Hlavní okno simulátoru. **Popis :** termostat je vypnutý, aktuálně neprobíhá žádné měření, nastaven je jednotlivý průběh měření. Tlačítko "Show more"zobrazí podrobné informace, které jsou na dalším obrázku.

[8] dále jen GUI

7.1 Grafická knihovna

Pro realizaci grafického uživatelského rozhraní byla zvolena knihovna GTK+, vhodná pro programování v C či C++. Knihovna umožňuje práci s aplikacemi, které používají Window Form. Obsahuje již přednastavené komponenty Widgety jako jsou tlačítka, textová pole, obrázky, dialogová okna apod. Widgety mají předprogramované události jako jsou stisk tlačítka, vložení textu, ukončení celé aplikace apod., pomocí kterých lze aplikaci snadno ovládat. Původní aplikace byla napsána v jazyku C. Proto bylo grafické rozhraní naprogramováno bez použití externích nástrojů pro tvorbu Window Form aplikací. V literatuře jsou uvedeny odkazy na internetové stránky, které obsahují podrobnější informace o knihovně GTK+[5][6].

7.2 Postup programování grafického rozhraní

Byl vyvořen seznam, kde byla uvedena všechna data, která je nutné zobrazit. Vybrala se ty nejdůležitější a byla rozmístěna do hlavního okna. Zbytek údajů byl rozmístěn do rozšířené části hlavního okna. Takto vzniklo první spustitelné okno aplikace.

Tímto seznamem byla také definována vnitřní struktura používaných proměnných v GUI. Ke každému atributu byly dopsány ovládací funkce a také nezbytné provázání se skutečným programem teploměru. Na straně GUI byli vygenerovány události, na základě kterých dochází ke změnám v teploměru. Stejně tak na straně teploměru vznikly události, na které musí zase reagovat GUI. Takto došlo k přehlednému určení všech ovládacích prvků GUI a snadnější realizaci.

Komunikace mezi GUI a teploměrem tedy probíhá opět pomocí událostí. Aplikace má k dispozici všechna aktuální nastavení teploměru (ke čtení). Teploměr má také k dispozici aktuální nastavení GUI - aktuální teplotu a adresu zařízení (opět pouze ke čtení). Tímto nedochází ke zbytečným duplikacím vnitřních dat a zároveň je zaručena jedinečnost dat.

Události generované GUI, na které reaguje třetí vrstva - digitální teploměr, jsou dvě :

- **gui_change_temp** - změna aktuální teploty teploměru,
- **gui_change_device_number** - změna adresy (čísla) zařízení digitálního teploměru.

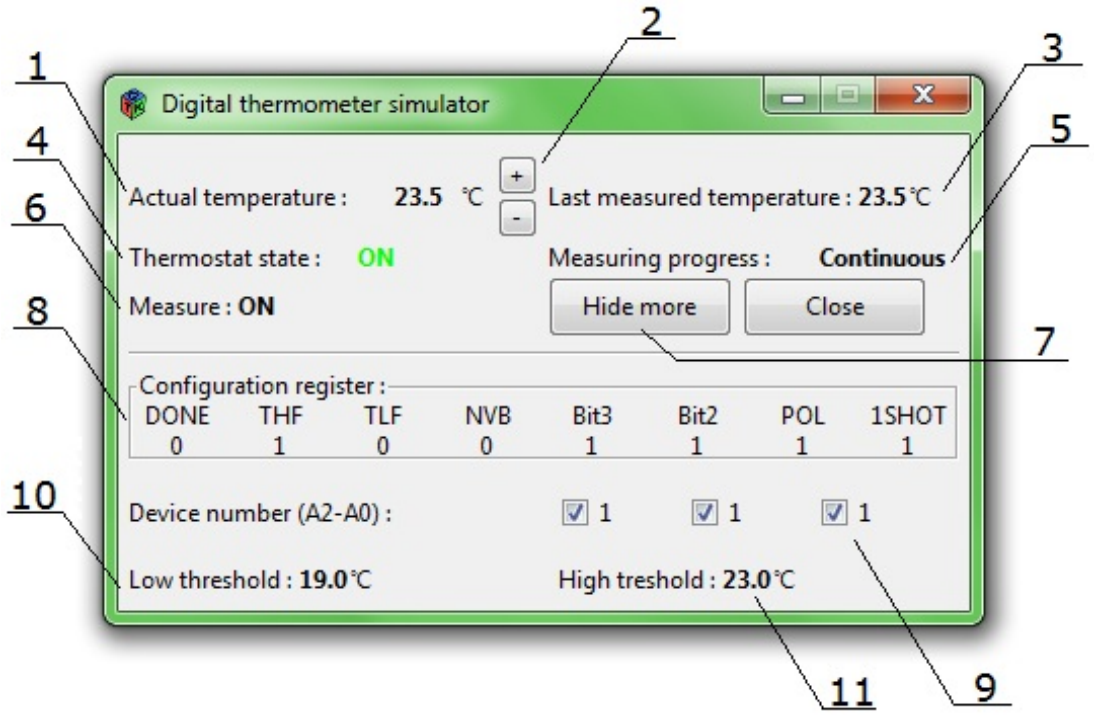
Události generované třetí vrstvou, na které reaguje GUI :

- **dev_change_temp** - změna poslední naměřené teploty,
- **dev_term_state** - změna ukazatele právě probíhajícího měření teploměru,
- **dev_state** - změna stavu termostatu,
- **dev_config** - změna konfiguračního registru,
- **dev_change_device_number** - změna adresy (čísla) zařízení digitálního teploměru,
- **dev_change_low** - změna dolního limitu teploměru,
- **dev_change_high** - změna horního limitu teploměru.

7.3 Podrobný popis GUI a komponent

Grafické uživatelské rozhraní bylo navrženo pro přehledné a jednoduché ovládání a monitorování simulátoru digitálního teploměru. Uživatel zde může nastavit aktuální teplotu teploměru pomocí tlačítek (+, -). Teplota se zde počítá po 0.5 stupních. Dále pak může uživatel změnit adresu volaného zařízení pomocí zaškrtávacích tlačítek. Na uvedených obrázcích č. 14, 15 a 16 je popis jednotlivých polí simulátoru a ukázka nastavení termostatu.

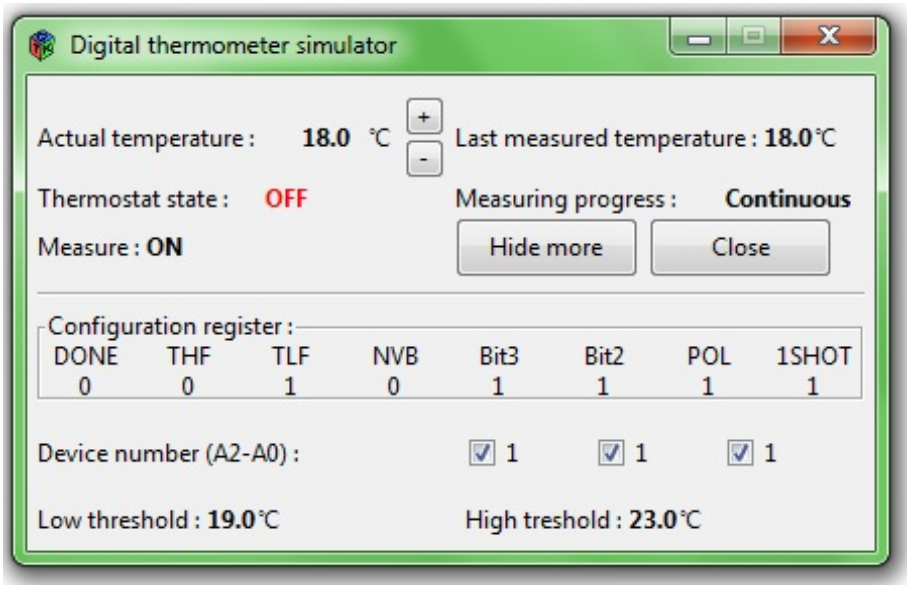
Aplikace simulátoru je také připravena pro případnou lokalizaci do jiných jazyků a má přednastavené knihovny pro případné použití pod operačním systémem Linux.



Obrázek 15: Hlavní okno simulátoru s info.

Popis : termostat je zapnutý a právě probíhá měření, je nastaven spojitý průběh měření, v konfiguračním registru je zaznamenáno také překročení horního limitu (pole THF je 1), dále je zde zaznamenána polarizace "active high"(pole POL je 1).

Popis : 1 - aktuální teplota, **2** - tlačítka pro změnu teploty, **3** - poslední naměřená teplota, **4** - stav termostatu (On/Off), **5** - průběh měření (Continous/ Single), **6** - měření probíhá (On/Off), **7** - tlačítko pro návrat do základního okna, **8** - výpis konfiguračního registru, **9** - adresa teploměru, **10** - dolní teplotní limit, **11** - horní teplotní limit.



Obrázek 16: Hlavní okno simulátoru s info.
Popis : Termostat je vypnutý, spojité měření stále probíhá, v konfiguračním registru je zaznamenáno překročení dolního limitu teploměru (pole TLF je 1).

8 Komunikační protokol klient-server

Synchronizace mezi GUI a digitálním teploměrem je zajištěna pomocí vláken a socketu.

Aplikace je založena na architektuře klient-server. Simulátor zde plní úlohu serveru a klientem se stává uživatelský program. Uživatel tedy může přes komunikační port ovládat činnost teploměru, dle vlastních požadavků. Zároveň je aplikace simulátoru na běhu klientské aplikace nezávislá a lze ji nastavovat i nezávisle na klientovi.

Komunikace mezi klientem a serverem je vedena přes socket. Ten zajistí její synchronizaci a umožňuje bezpečně zasílat data na server a zpět. Data, která se zasílají, jsou jen dvou bitová, reprezentují aktuální stav linek SCL a SDA. Samotné posílání dat je prováděno pomocí zpráv.

Každá zpráva má délků maximálně osm znaků včetně ukončovací nuly '\0'. Prvních pět znaků zprávy označuje příkaz a další dva zasílaná data.

Zprávy odesílané klientem :

SDA__0# - zapisuje na sběrnici data z linky SDA

SCL__0# - zapisuje na sběrnici data z linky SCL

get__ - požaduje zaslání aktuálního stavu linek SCL a SDA

Zprávy odesílané serverem :

connect – ohlašuje úspěšné připojení klienta k simulátoru

get__## - zasílá klientovi aktuální stav linek SCL (1. číslo) a SDA (2. číslo)

decline - odmítnutí klienta

Funkcionalita použitého socketu a vláken byla převzata z diplomové práce pana Michala Fůse[3]. Jedná se jak o stranu klienta, tak i serveru. Dopracováno bylo zasílání zpráv obsahující data ke zpracování ze strany serveru ke klientovi. Nově přidané typy zpráv jsou tedy get__ ze strany klienta a get__## ze strany serveru.

Současně se také změnila funkce set__##, která měla původně sloužit k nastavení SDA i SCL linek současně. Ovšem při testování bylo zjištěno, že klient nemá vždy aktuální informace o stavu obou linek, takže docházelo k nežádanému přepisování hodnot na lince SDA. Jedním z řešení by bylo se vždy dotázat před každým novým nastavením jedné z linek na aktuální stav druhé linky. To by ovšem mělo za následek zaslání dalších dvou zpráv get. Proto byla původní funkce set__## nahrazena funkcemi SDA__0# a SCL__0#. Tímto je vždy nastavena pouze změna na jedné z linek a nedochází k přepisu druhé linky, která mezitím mohla změnit svou hodnotu. Původní funkce určené pro obvod digitálního teploměru pro zápis a čtení dat z linek SDA a SCL jsou uvedeny v kapitole 6. Přepracované funkce, využívající socket, jsou uvedeny v příloze B.1.

8.1 Práce s vlákny aplikace

Ve funkci **int** main(**int** argc, **char** ∗argv[]) simulátoru se spouští dvě vlákna, jedno je určeno pro aplikaci GUI a druhé pro klienta jako obslužné vlákno. Synchronizace a řízení je prováděno pomocí globálních proměnných, zámků a signálů.

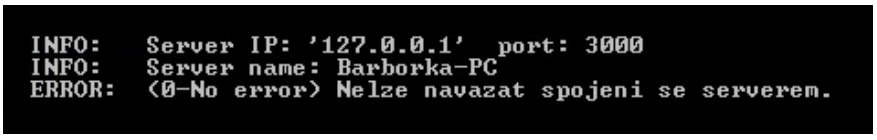
Po spuštění aplikace a její inicializaci se vytvoří samostatné vlákno s GUI. V této chvíli inicializuje GUI své proměnné a vytváří hlavní okno aplikace. Dále se spouští druhé vlákno, určené pro obsluhu klienta. Nakonec se inicializuje samotný socket pomocí funkce init_socket(**int** port). Port **int** port = 3000 pro aplikaci simulátoru zůstává neměnný. [3]Socket je v aplikaci vytvářen voláním funkce socket(**int** socket_family, **int** socket_type, **int** protocol) s následujícím nastavením :

AF_NET - bude použit Internet Protocol v. 4 (IPv4)

SOCK_STREAM - určuje, že bude použit TCP protokol, který zaručuje spolehlivou obousměrnou komunikaci.

Poté se pomocí funkce bind přiřadí adresa a port socketu. Po úspěšném přiřazení aplikace naslouchá na zadaném portu a očekává připojení klienta. Jakmile se klient připojí, zašle mu zprávu connect[9] a předává řízení obslužnému vláknu. Pokud je k simulátoru již nějaký klient připojen, pak je nový klient odmítnut se zprávou decline a spojení s ním je ukončeno[10].

Obrázek 17: Ukázka spojení klienta se simulátorem.



Obrázek 18: Ukázka nevydařeného spojení klienta se simulátorem.

[9] viz. obrázek č. 17
[10] viz. obrázek č. 18

9 Vyhodnocení spolehlivosti a stability

9.1 Naplánované testy aplikace

Byla provedena série testů ke zjištění stability a spolehlivosti naprogramovaného simulátoru. V rámci testování bylo porovnáno chování simulátoru a skutečné fyzické součástky obvodu digitálního teploměru. Pro jednodušší a rychlejší průběh testů bylo vytvořeno pět pomocných funkcí, které reprezentují pět hlavních typů komunikace viz. kapitola 4.2.1. Tyto funkce jsou uvedeny v příloze B.2.

Seznam provedených testů chování při běžném užívání (srovnání s fyzickou součástkou) :

- zkouška spojení s teploměrem,
- zjištění reakce po zadání adresy teploměru,
- otestování všech používaných příkazů,
- otestování reakce GUI na změny :
  - konfiguračního registru,
  - horní a dolní hranice termostatu,
  - jednotlivé stavy termostatu - zapnutí, vypnutí, způsob měření, probíhající měření,
  - poslední naměřené teploty,
- zkouška připojení více klientů,
- otestování paměti teploměru, zda po ukončení připojení klienta udržuje aktuální nastavení termostatu,
- otestování správného čtení a zápisu dat do konfiguračního souboru aplikace.

Seznam provedených testů stability aplikace :

- nastavení extrémních hodnot do horní a dolní hranice termostatu
- zadání neexistující adresy zařízení
- zadání neexistujícího příkazu
- neukončení komunikace s teploměrem
- přihlášení nového klienta s tím, že předchozí klient neuzavřel spojení
- pokus o zaslání více než jednoho bajtu najednou

9.2 Výsledky testů

9.2.1 Testy chování při běžném užívání

Klient ve všech případech úspěšně navázal spojení se simulátorem.

Po vložení správné adresy potvrdil její přijetí příkazem ACK.

Teploměr rozeznal všechny používané příkazy. V pořádku přijal všechna očekávaná data a odeslal všechna požadovaná data.

GUI správně reagovalo na všechny změny v konfiguračním registru teploměru. Dále také správně nastavilo změněné horní i dolní limity termostatu. Při jednotlivém i spojitém měření správně měnilo poslední naměřenou teplotu. Termostat se zapínal i vypínal ve správnou chvíli, všechny stavy reagovali očekávaným způsobem.

Při pokusu o přihlášení klienta, kdy ve stejnou dobu již nějaký klient je připojen k simulátoru, byl nově příchozí klient odmítnut.

Po ukončení připojení klienta bylo zachováno aktuální nastavení aplikace simulátoru.

Při spuštění aplikace simulátor přečte a nastaví všechny údaje z konfiguračního souboru. Při ukončení aplikace je opět správně zapíše zpět do konfiguračního souboru.

9.2.2 Testy stability aplikace

Při pokusu o nastavení extrémní či nevyhovující hodnoty do horní či dolní hranice termostatu, reaguje simulátor tak, že novou hodnotu nastaví. Pokud hodnota přesahuje velikost jednoho bajtu, pak je uložen jen jeden bajt. Při nesprávném nastavení termostatu je však jeho chování nepředvídatelné.

Pokud klient zadá neexistující adresu zařízení, teploměr neodpoví a tudíž žádná další komunikace se zařízením není možná, dokud klient neodešle na sběrnici správnou adresu. Termostat nereaguje na jakoukoliv komunikaci.

Stejně tak, pokud se klient pokusí zaslat neexistující příkaz, pak jakákoliv další data, která teploměr obdrží nejsou přijímána ani zpracována.

Pokud klient neukončí komunikaci s teploměrem příkazem STOP, pak teploměr po ukončení klientova spojení, sám rozpozná konec komunikace. Proto přihlášení nového klienta nemá žádné problémy.

A nakonec, pokud se klient snaží poslat více než jeden bajt najednou, zašle se na sběrnici vždy jen jeden bajt.

9.3 Vyhodnocení testů

Na závěr bylo provedeno srovnání chování a ovládání fyzické součástky digitálního teploměru a jeho programové simulace. Po „naprogramování“ je chování totožné.

Je důležité podotknout, že v původní podobě si student musel veškeré získané údaje z teploměru vypisovat na konzoli či jinak ukládat, nyní má veškeré potřebné údaje k dispozici hned a jsou neustále aktualizovány. Pro případné vypisování dat na konzoli může využít systém zpráv, který je k dispozici v knihovnách klienta. Práce s programovým simulátorem bude pro studenty srovnatelná.

10 Závěr

Programový simulátor obvodu digitálního teploměru má k dipozici všechny požadované funkce používané při výuce. V GUI lze nastavovat aktuální teplotu a případně měnit adresu volaného zařízení. Studenti nyní mohou nastavit obvodu požadovanou tří bitovou adresu a programově ověřit, zda je obvod na zadané adrese. Dále pak lze nastavit teploměr pro spojitý převod teploty, spustit převod teploty a číst aktuální teplotu. V neposlední řadě lze nastavit obvod teploměru jako termostat. Všechny příkazy pro digitální teploměr jsou plně funkční. Studenti mohou číst data teploměru či naopak data zapisovat a nastavovat jeho vlastnosti. Po ukončení práce s aplikací je aktuální nastavení teploměru uloženo do konfiguračního souboru, ze kterého se při dalším spuštění aplikace opět nastaví do posledního známého stavu.

Aplikace simulátoru je spustitelná v operačním systému Windows. Přenositelnost pro operační systém Linux by měla být bezproblémová, v programu jsou knihovny používané pro Linux již naimplementovány.

Aplikace je také připravena pro případnou lokalizaci do jiných jazyků. Nyní je využívána pouze angličtina.

Podrobné informace o digitálním teploměru, I^2C sběrnici či o předmětu APPS najdete v publikacích uvedených v literatuře a v přílohách.

Barbora Švajcová

11 Reference

[1] Jean-Marc Irazabal, Steve Blozis, *AN10216-01 I^2C Manual* [online]. 2003. [cit. 2013-21-09].
<http://www.nxp.com/documents/application_note/AN10216.pdf>

[2] *DS1621 Digital Thermometer and Thermostat*. Datový list digitálního teploměru. [online]. 2012. [cit. 2013-09-09].
<http://poli.cs.vsb.cz/edu/apps/lab/DS1621.pdf>

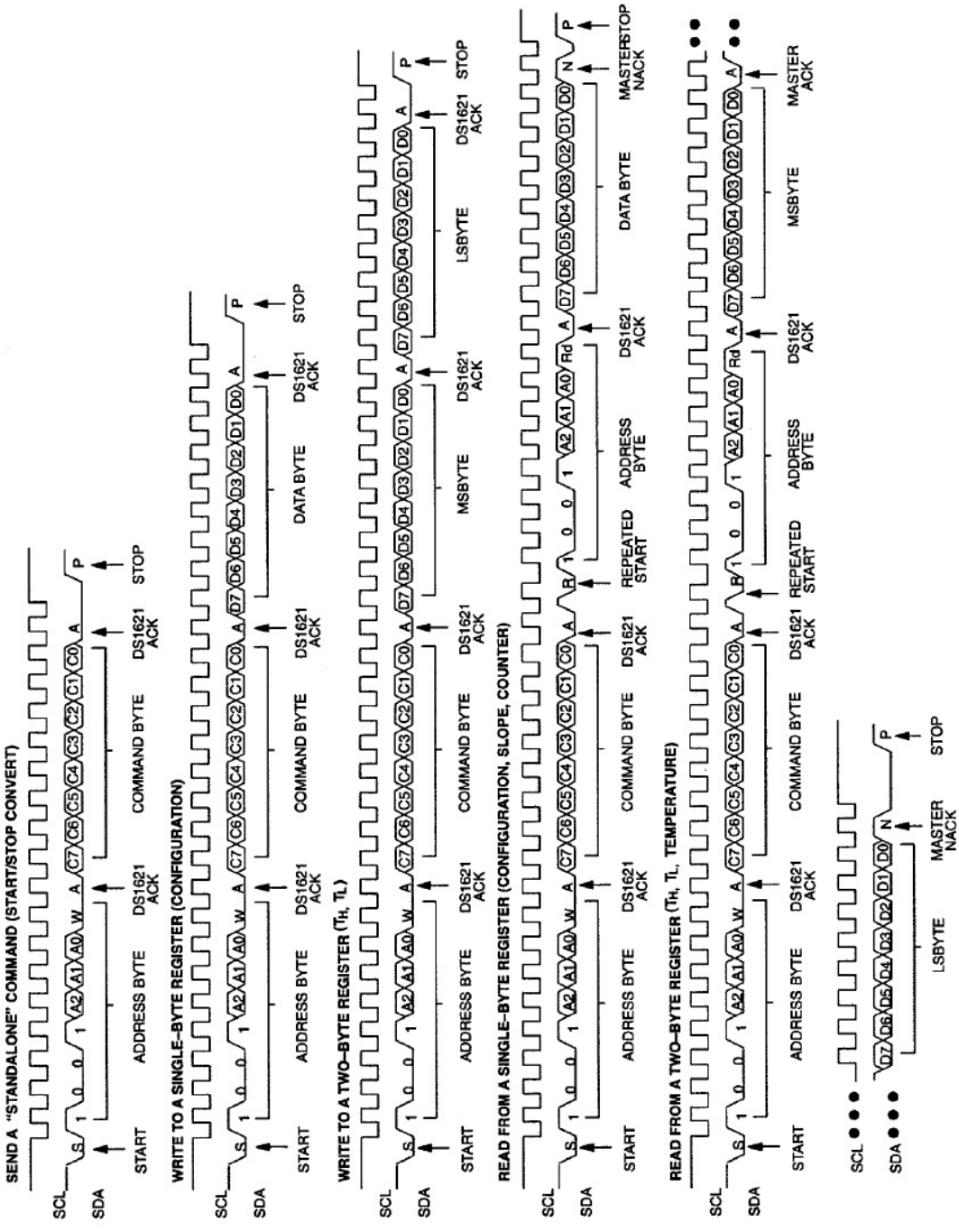
[3] Michal Fůs, *Simulace LED řízených PWM jako aplikace v GUI*, Ostrava, 2013. 49s. Diplomová práce. VŠB-TU Ostrava, FEI.

[4] Petr Olivka, David Seidl, *Architektury počítačů a paralelních systémů,* [online], 2011. [cit. 2013-08-14]. VŠB-TU Ostrava, FEI.
<http://poli.cs.vsb.cz/edu/apps/lab/apps-cvic.pdf>

[5] Gnome Developer, *GTK+ 2 Reference Manual* [online]. [cit. 2013-12-12].
<https://developer.gnome.org/gtk2/2.24/>

[6] Tony Gale, Ian Main and the GTK team, *GTK+ 2.0 Tutorial* [online]. [cit. 2013-05-12].
<http://oss.org.cn/man/develop/GTK+/tutorial/>

A Teploměr Dallas DS1621



Obrázek 19: Komunikace teploměru DS1621 se zařízením master[2].

B Zdrojové kódy

B.1 Zápis a čtení dat pomocí socketu

```
void I2C_SDA( char value ) // nastaveni signalu SDA
{
    Set_SDA(value);
}

char I2C_getSDA( void ) // cteni signalu SDA
{
     return Get_SDA();
}

void I2C_SCL( char value ) // nastaveni signalu SCL
{
    Set_SCL(value);
}


int Set_SDA( unsigned char value )
{
  if (value > 0)
  {
      SDA = 1;
  }
  else
  {
      SDA = 0;
  }
  char buf[8];
  sprintf (buf,"SDA__0%d", SDA );
  int r = WriteSock( sock_server, buf, MAX_LENGTH + 1 );
  if (r <= 0)
  {
    msg(MSG_ERR, "Neuspesne␣odeslano:␣%s", buf);
    return -1;
  }
  else
  {
    msg(MSG_DBG, "Uspesne␣odeslano:␣%s", buf);
    return 0;
  }
}


int Set_SCL( unsigned char value )
{
  if (value > 0)
  {
      SCL = 1;
  }
  else
```

```
    {
        SCL = 0;
    }
    char buf[8];
    sprintf (buf,"SCL__0%d", SDA );
    int r = WriteSock( sock_server, buf, MAX_LENGTH + 1 );
    if (r <= 0)
    {
      msg(MSG_ERR, "Neuspesne␣odeslano:␣%s", buf);
      return -1;
    }
    else
    {
      msg(MSG_DBG, "Uspesne␣odeslano:␣%s", buf);
      return 0;
    }
}

char Get_SDA ()
{
        sprintf (buf, "get__");
        int r = WriteSock( sock_server, buf, MAX_LENGTH + 1 );
        if (r <= 0)
        {
          msg(MSG_ERR, "Neuspesne␣odeslano:␣%s", buf);
          return -1;
        }
        else
        {
            r = ReadSock( sock_server, buf, MAX_LENGTH + 1);
            msg(MSG_DBG, "Prijato:␣%s", buf);
            if (r < 0)
            {
              msg( MSG_ERR, "precteni␣hodnoty␣SDA␣selhalo");
              return -1;
            }
            if (!strncasecmp(buf,"get__", 5 ))
            {
              char data = ( strtol (buf + 5, NULL, 2));
              SDA = (data & 0x1);
              msg(MSG_DBG, "Precteno␣SDA␣:␣%d", SDA);
              return SDA;
            }
            else
            {
              msg( MSG_ERR, "precteni␣hodnoty␣SDA␣selhalo");
              return -1;
            }
        }
    return -1;
}
```

Výpis 5: Zápis a čtení dat ze socketu - strana klienta.

```
void *service_thread(void)
{
    ....
    //prijem dat od klienta
     int r = ReadSock( client_socket, buf, MAX_LENGTH + 1 ) ;
    ...

    //obsluha klienta
    if (!strncasecmp(buf,"SDA__",CMDLEN))
    {
       Bus_Set_SDA(strtol(buf + CMDLEN, NULL, 2));
    }
    else if (!strncasecmp(buf,"SCL__",CMDLEN))
    {
       Bus_Set_SCL(strtol(buf + CMDLEN, NULL, 2));
    }

    else if (!strncasecmp(buf,"get__",CMDLEN))
    {
        sprintf ( buf, "get__%d%d", Bus_Get_SCL(), Bus_Get_SDA());
       int r = WriteSock(client_socket,buf,MAX_LENGTH + 1);
       if (r <= 0)
       {
            msg(MSG_ERR, "Neuspesne␣odeslano:␣%s", buf);
       }
       else
       {
          msg(MSG_DBG, "Uspesne␣odeslano:␣%s}", buf);
       }
    }
    ...
}
```

Výpis 6: Zápis a čtení dat ze socketu - strana serveru.

B.2 Testovací funkce komunikace zařízení master a slave

```
void Command_StandAlone(unsigned char address, unsigned char command);
void Command_WriteSingleByte(unsigned char address, unsigned char command,
                             unsigned char dataByte);
void Command_WriteTwoByte(unsigned char address, unsigned char command,
                          unsigned char msByte, unsigned char lsByte);
void Command_ReadSingleByte(unsigned char address, unsigned char command);
void Command_ReadTwoByte(unsigned char address, unsigned char command);

char ack;
char *err_cmdByte = "CHYBA > Neprobehlo, nebyl prijat command byte\n";
char *err_msbByte = "CHYBA > Neprobehlo, nebyl prijat byte MSB\n";
char *err_address = "CHYBA > Neprobehlo, nenalezeno volane zarizeni\n";
char *err_lsbByte = "CHYBA > Neprobehlo, nebyl prijat posledni byte\n";
char *ok = " .... probehlo uspesne\n";
int main()
{
  // inicializace klienta
  I2CPort_init () ;
  // uvedeni sbernice do klidoveho stavu
  I2C_Init () ;

  unsigned char address = 0b10011110;
  unsigned char config = 0b10000011;

  Command_WriteSingleByte(address, 0xAC, config); // zapis do konfig. registru
  Command_StandAlone(address, 0xEE); // start convert

  I2C_Stop();
  system("PAUSE");
  return 0;
}

void Command_StandAlone(unsigned char address, unsigned char command)
{
  printf ("STAND ALONE\n");
  I2C_Start();
  // printf ("      P R V N I   V Y S T U P   A D R E S A\n");
  ack = I2C_Vystup( address ); // adresa vcetne R_W
  if (!ack)
  {
    // printf ("     D R U H Y  V Y S T U P\n");
    ack = I2C_Vystup( command ); // command byte
    if (!ack)
    {
      printf ("%s",ok);
    }
    else { printf ("%s", err_cmdByte); }
  }
  else { printf ("%s", err_address); }
}
```

```
void Command_WriteSingleByte(unsigned char address, unsigned char command, unsigned
      char dataByte)
{
   printf ("WRITE␣SINGLE␣BYTE\n");
  I2C_Start();
  // printf ("       P R V N I    V Y S T U P   A D R E S A\n");
  ack = I2C_Vystup( address ); // adresa vcetne R_W
   if  (!ack)
   {
     // printf ("      D R U H Y   V Y S T U P \n");
     ack = I2C_Vystup( command ); // command byte
      if  (!ack)
      {
        // printf ("        T R E T I    V Y S T U P   D A T A B Y T E\n");
        ack = I2C_Vystup( dataByte );


        if  (!ack)
        {
             printf ("%s",ok);
        }
        else {  printf ("%s", err_lsbByte);  }
      }
     else {  printf ("%s", err_cmdByte); }
   }
   else {  printf ("%s", err_address); }
}

void Command_WriteTwoByte(unsigned char address, unsigned char command, unsigned char
      msByte, unsigned char lsByte)
{
   printf ("WRITE␣TWO␣BYTE\n");
  I2C_Start();
  // printf ("       P R V N I    V Y S T U P   A D R E S A\n");
  ack = I2C_Vystup( address ); // adresa vcetne R_W
   if  (!ack)
   {
     // printf ("      D R U H Y   V Y S T U P \n");
     ack = I2C_Vystup( command ); // command byte
      if  (!ack)
      {
        // printf ("        T R E T I    V Y S T U P   M S B Y T E\n");
        ack = I2C_Vystup( msByte );
        if  (!ack)
        {
          // printf ("   C T V R T Y   V Y S T U P   L S B Y T E\n");
          ack = I2C_Vystup( lsByte );
          if  (!ack)
          {
               printf ("%s", ok);
          }
          else {  printf ("%s", err_lsbByte);  }
        }
        else {  printf ("%s", err_msbByte); }
```

```
    }
    else {  printf ("%s", err_cmdByte); }
  }
  else {  printf ("%s", err_address); }
}

void Command_ReadSingleByte(unsigned char address, unsigned char command)
{
  unsigned char addrRead = address;
   printf ("READ␣SINGLE␣BYTE\n");
  I2C_Start();
  // printf ("       P R V N I   V Y S T U P   A D R E S A\n");
  ack = I2C_Vystup( address ); // adresa vcetne R_W
  if  (!ack)
  {
    // printf ("      D R U H Y   V Y S T U P \n");
    ack = I2C_Vystup( command ); // command byte
    I2C_Start();  //REPEAT
    addrRead |= 1; // nastaveni read
    if  (!ack)
    {
      // printf ("       T R E T I   V Y S T U P   O P A K O V A N I   A D R E S Y\n");
      ack = I2C_Vystup( addrRead );
      if  (!ack)
      {
        unsigned char vstup = I2C_Vstup();
         printf ("Byte␣obdrzeny␣od␣slave␣0x%X\n", vstup);
         // master NACK
        I2C_NAck();
      }
      else {  printf ("%s␣repeat\n", err_address); }
    }
    else {  printf (err_cmdByte); }
  }
  else {  printf (err_address); }
}

void Command_ReadTwoByte(unsigned char address, unsigned char command)
{
  unsigned char addrRead = address;
   printf ("READ␣TWO␣BYTE\n");
  I2C_Start();
  // printf ("       P R V N I   V Y S T U P   A D R E S A\n");
  char ack = I2C_Vystup( address ); // adresa vcetne R_W
  if  (!ack)
  {
    // printf ("      D R U H Y   V Y S T U P \n");
    ack = I2C_Vystup( command ); // command byte
    I2C_Start();   //REPEAT
    addrRead |= 1; // nastaveni read
    if  (!ack)
    {
      // printf ("       T R E T I   V Y S T U P   O P A K O V A N I   A D R E S Y\n");
      ack = I2C_Vystup( addrRead );
```

```
        if (!ack)
        {
          // printf ("          P R V N I   V S T U P \n");
          unsigned char vstup = I2C_Vstup();
           printf ("MSB byte od slave 0x%X ", vstup);
           if (vstup != "")
           {
                I2C_Ack();
                // printf ("          D R U H Y   V S T U P \n");
                vstup = I2C_Vstup();
                 printf ("LSB byte 0x%x\n", vstup);
                I2C_NAck();
           }
           else
           {
                I2C_NAck();
           }
        }
        else { printf ("%s repeat addr", err_address); }
      }
      else { printf (err_cmdByte); }
    }
    else { printf (err_address); }
}
```

Výpis 7: Pomocné funkce pro jednotlivé druhy komunikace master a slave zařízení.

C Příloha na CD

Adresářová struktura přiloženého CD :

- **Software**
  - **I2C_Client** - CodeBlock projekt klienta (použitá testovací verze).
  - **Simulator** - CodeBlock projekt simulátoru.
    - **Win** - knihovny a zdrojové soubory třetích stran.
- **Ukázky**
  - **I2C_Client** - spustitelné soubory s programovými nastaveními klienta.
  - **Simulator** - aplikace simulátoru.
- **Text** - text diplomové práce.